Institut für Numerische Simulation der Universität Bonn
Prof. Dr. Mario Bebendorf

Praktikum im Sommersemester 2013

Programmierpraktikum numerische Algorithmen (P2E1)
(Numerische Lösung der Wärmeleitungsgleichung)

Betreuer: Christian Kuske[1]

Blatt 3

1 Einführung

Die Aufgaben von Blatt 2 haben gezeigt, dass sich das Gauss-Seidel-Verfahren durch eine Überrelaxation verbessern lässt. Trotzdem ist die Verwendung des CG-Verfahrens für allgemeinere Probleme besser geeignet, da die Berechnung eines optimalen Relaxationsparameters entfällt. Des Weiteren wurden Vorkonditionierer vorgestellt, die die Anzahl der CG-Schritte reduzieren.

Teil dieses Aufgabenblatts wird es sein, die auf dem letzten Blatt erwähnte IC-Zerlegung als Vorkonditionierer für den diskretisierten zweidimensionalen Laplace-Operator zu implementieren. Hierbei kann es besonders für große Gleichungssysteme von Vorteil sein, die Vorkonditionierung blockweise durchzuführen und somit jeden Teil der Matrix unterschiedlich zu behandeln und dadurch eine Parallelisierung zu ermöglichen.

2 ILU-Zerlegung

Die unvollständige LU-Zerlegung führt bei schwach-besetzten Gleichungssystemen, auf einen brauchbaren Vorkonditionierer für das CG-Verfahren. Diese Methode lässt sich damit ohne genaue Kenntnis der Einträge beziehungsweise der genauen Struktur der Matrix einsetzen und ermöglicht in den meisten Fällen eine Verringerung der Schrittzahl. Der folgende Algorithmus beschreibt die ILU-Zerlegung, die im Vergleich zur LU-Zerlegung eine zusätzliche Menge Z benötigt, die die zu besetzenden Stellen angibt.

```
for k = 1 : n − 1 do
  for i = k + 1 : n do
    if (i, k) ∈ Z then a_ik := a_ik / a_kk
    for j = k + 1 : n do
      if (i, j) ∈ Z then a_ij := a_ij − a_ik a_kj
```

Algorithmus 2.1: ILU-Zerlegung

[1]We6 4.002; Tel.: 0228/73-60454; Email: kuske@ins.uni-bonn.de

Ohne Fill-In

Die einfachste Version der ILU-Zerlegung besteht darin, die Einträge der Matrizen L und U nur für die Stellen zu berechnen an denen A ungleich Null ist. In diesem Fall ist $Z := \{(i,j) \,|\, a_{ij} \neq 0\}$. Die ILU-Zerlegung stimmt an $(i,j) \in Z$ mit der LU-Zerlegung überein. Alle anderen Stellen in L beziehungsweise U werden null gesetzt.

Mit Fill-In

Hierbei werden bei der Zerlegung zusätzliche Einträge der Matrix A als ungleich Null betrachtet. Dadurch wird die Menge Z durch zusätzliche Indizes vergrößert und die Approximation an die Matrix A genauer. Die Anzahl der CG-Schritte kann so verringert werden.

Zur Realisierung dieser Methode wird mittels einer Funktion jedem Eintrag der Matrix A ein Fill-In-Niveau zugeordnet. Dabei haben die Einträge, in denen A ungleich Null ist, ein Fill-In-Niveau von Null. Das bedeutet, dass diese Einträge in jedem Fall verwendet werden. Für alle restlichen Einträge können unterschiedliche Strategien gewählt werden, die die Struktur der Matrix, den Rechenaufwand oder die Genauigkeit der Approximation bevorzugen.

Mit Struktur

Eine weitere Möglichkeit ist, die Einträge der Matrizen L und U ungeachtet der Struktur der Matrix A festzulegen. Das heißt, dass die Menge Z bestimmte Indizes beinhaltet, die unabhängig von A sein können. Dabei können beispielsweise numerische Gesichtspunkte für eine möglichst einfache Struktur in den Vordergrund gestellt werden.

3 IC-Zerlegung

Die unvollständige Cholesky-Zerlegung kann ähnlich wie die ILU-Zerlegung erstellt werden. Hierfür muss darauf geachtet werden, dass lediglich die Berechnung der unteren Dreiecksmatrix nötig ist. Das resultierende Verfahren ist gegenüber der ILU-Zerlegung numerisch stabiler, kann allerdings lediglich für symmetrische Matrizen verwendet werden. Der Algorithmus hat die folgenden Gestalt:

```
for k = 1 : n do
  a_kk := √a_kk
  for i = k + 1 : n do
    if (i,k) ∈ Z then a_ik := a_ik / a_kk
  for j = k + 1 : n do
    for i = j : n do
      if (i,j) ∈ Z then a_ij := a_ij − a_ik a_jk
```

Algorithmus 3.1: IC-Zerlegung

Die Bedingungen an die Menge Z sind die gleichen wie bei der ILU-Zerlegung beschrieben. Dabei hängt die optimale Wahl der Elemente in Z von der Problemstellung ab. Allerdings führt eine Wahl ohne Fill-In bei den meisten Problemstellungen bereits zu einer deutlichen Beschleunigung des CG-Verfahrens.

4 Block-Zerlegung

Bei strukturierten oder großen Matrizen kann eine Zerlegung in kleinere Teilmatrizen Laufzeit und Speicheraufwand reduzieren. Hierbei lassen sich die Block-Zerlegungen für direkte Verfahren

(LU- oder Cholesky-Zerlegung), iterative Verfahren (Gauss-Seidel, Jacobi oder SOR) oder Vorkonditionierer verwenden. Der Vorteil ergibt sich durch das Aufteilen in kleinere Probleme, die individuell behandelt und gegebenenfalls parallel berechnet werden können.

Direkte Verfahren

Das Vorgehen der Block-LU-Zerlegung entspricht dem der LU-Zerlegung mit dem Unterschied, dass Matrizen statt Skalaren verwendet werden. Ist eine Matrix A beispielsweise in vier Blöcke geteilt, so ergibt sich folgende Darstellung:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \cdot \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix},$$

wobei die Matrizen L_{ii}, U_{ii} Dreiecksmatrizen und A_{ij}, L_{ij}, U_{ij} von geeigneter Größe sind.

Zunächst wird die LU-Zerlegung von A_{11} in $L_{11}U_{11}$ bestimmt. Im zweiten Schritt folgen die Berechnungen mittels Vorwärts- beziehungsweise Rückwärtssubstitution von U_{12} aus $L_{11}U_{12} = A_{12}$ beziehungsweise L_{21} aus $L_{21}U_{11} = A_{21}$. Als letztes wird die LU-Zerlegung von $A_{22} - L_{21}U_{12}$ in $L_{22}U_{22}$ bestimmt.

Diese Zerlegung lässt sich rekursiv fortsetzen, sodass in der letzten Zerlegung die Matrizen lediglich die Größe 1×1 besitzen. Aus laufzeittechnischen Gründen ist es allerdings sinnvoller, die Rekursion bereits vorher zu stoppen.

Auf analoge Weise ergibt sich für symmetrische Matrizen die Block-Cholesky-Zerlegung.

Iterations-Verfahren

Die Block-Jacobi-Iteration lässt sich analog zur Jacobi-Iteration knstruieren. Dazu wird die Matrix A in $D - L - U$ zerlegt, wobei D eine Block-Diagonalmatrix aus den Diagonalblöcken aus A und L, U die Block-Dreiecksmatrizen sind. Damit ergibt sich die Iterationsvorschrift

$$A_{jj}x_j^{(k+1)} = b_j - \sum_{i \neq j} A_{ji}x_i^{(k)}.$$

Hierbei müssen in jedem Schritt von $x^{(k)}$ nach $x^{(k+1)}$ lineare Gleichungssysteme gelöst werden. Für diese kleineren Gleichungssysteme können LU- oder Cholesky-Zerlegung bestimmt werden. Die Blockmatrizen A_{ij} sollten eine Struktur aufweisen, die sich schnell invertieren lässt. Beispielsweise besitzt der diskretisierte Laplace-Operator auf den Diagonalblöcken A_{ii} eine solche Struktur.

Auch Gauss-Seidel- und SOR-Iteration lassen sich durch eine ähnliche Vorgehensweise konstruieren.

Vorkonditionierer

Die Idee der Blockmatrizen lässt sich ebenfalls für die Konstruktion von Vorkonditionierern nutzen. Hierbei können einzelne Blöcke unterschiedlich behandelt werden und dadurch eine bessere Approximation an die Inverse liefern. Der einfachste dieser Vorkonditionierer ist wie im skalaren Fall der Block-Jacobi-Vorkonditionierer. Hierbei werden statt der Diagonaleinträge die Diagonalblöcke invertiert und auf das entsprechende Gleichungssystem als Vorkonditionierer angewendet.

Dieses Vorgehen ist nicht auf das Jacobi-Verfahren beschränkt, sondern auch auf die auf Blatt 2 vorgestellten Vorkonditionierer anwendbar.

Parallelisierung

Die oben genannten Verfahren zur Verwendung von Blockmatrizen lassen es zu, gewisse Berechnungen parallel durchzuführen. Dabei ist zu beachten, dass Abhängigkeiten von Blöcken untereinander berücksichtigt werden.

5 Aufgaben

a) Implementiere das SSOR-Verfahren als Vorkonditionierer für das PCG-Verfahren und verwende folgende Funktion:

   - `void LaplaceMatrixSSOR::applyPrecond(double * const x) const`,
     wobei $x = Cx$ berechnet wird und C der Vonkonditioniere des SSOR-Verfahren ist. (Hinweis: Die Matrix C wird dabei nicht aufgestellt.)

b) Implementiere die IC-Zerlegung als Vorkonditionierer für das PCG-Verfahren und gehe wie folgt vor:

   – Erstelle die Matrix-Klasse `LaplaceMatrixIC` mit drei Vektoren `diag_, subdiag_, subsubdiag_` vom Typ `double * const`, so dass die Diagonale und die beiden Subdiagonalen der Vorkonditionierer-Matrix $C = (LL^T)^{-1}$ gespeichert werden können. (Hinweis: Der Speicher muss im Konstruktor erstellt und im Destruktor wieder freigegeben werden.)

   – `void LaplaceMatrixIC::initPrecond()`,
     wobei die Funktion die drei Vektoren `diag_, subdiag_, subsubdiag_` berechnet. Das Besetzungmuster von L entspricht dabei der unteren Dreiecksmatrix von A.

   – `void LaplaceMatrixIC::applyPrecond(double * const x)` const,
     wobei $x = Cx$ berechnet wird und $C = (LL^T)^{-1}$ der Vorkonditionierer der IC-Zerlegung ist. (Hinweis: Die Matrix C ist durch die drei Vektoren `diag_`, `subdiag_`, `subsubdiag_` gegeben und durch Vorwärts- und Rückwärtssubstitution zu berechnen.)

c) Vergleiche Rechenzeiten und Iterationsschritte für CG- und PCG-Verfahren (SSOR- und IC-Vorkonditionierer) mit unterschiedlichen n und ε.